CertiCrypt
Language-Based Cryptographic Proofs in Coq

Gilles Barthe\textsuperscript{1,2} Benjamin Grégoire\textsuperscript{1,3}
Santiago Zanella\textsuperscript{1,3}

\textsuperscript{1}Microsoft Research - INRIA Joint Centre, France
\textsuperscript{2}IMDEA Software, Madrid, Spain
\textsuperscript{3}INRIA Sophia Antipolis - Méditerranée, France

POPL 2009
What’s wrong with cryptographic proofs?

- In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor
  M. Bellare and P. Rogaway.

- Do we have a problem with cryptographic proofs? Yes, we do [...] We generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect)
  S. Halevi

- Security proofs in cryptography may be organized as sequences of games [...] this can be a useful tool in taming the complexity of security proofs that might otherwise become so messy, complicated, and subtle as to be nearly impossible to verify
  V. Shoup
What’s wrong with cryptographic proofs?

- In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor. M. Bellare and P. Rogaway.

- Do we have a problem with cryptographic proofs? Yes, we do [...] We generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect). S. Halevi

- Security proofs in cryptography may be organized as sequences of games [...] this can be a useful tool in taming the complexity of security proofs that might otherwise become so messy, complicated, and subtle as to be nearly impossible to verify. V. Shoup
Game-based cryptographic proofs

Attack Game

\[ \text{Game } G_0^\eta : \]

\[ \ldots \]

\[ A \]

\[ \ldots \]

\[ \Pr_{G_0^\eta}[A_0] \leq \epsilon(\eta) \]

Security property
Game-based cryptographic proofs

\[ \Pr_{G_0^\eta}[A_0] \leq h_1(\Pr_{G_1^\eta}[A_1]) \leq \cdots \leq h_n(\Pr_{G_n^\eta}[A_n]) \]

\[ \Pr_{G_0^\eta}[A_0] \leq h(\Pr_{G_n^\eta}[A_n]) \leq \epsilon(\eta) \]
Game-based proofs: essence and problems

Independent events

\[ \Pr_{G_0}[A_0] \leq h(\Pr_{G}[A]) \times h'(\Pr_{G'}[A']) \]

Essence: relate the probability of events in consecutive games

But,

- How do we represent games?
- What adversaries are feasible?
- How do we make a proof hold for any feasible adversary?
Game-based proofs: essence and problems

Independent events

\[ \Pr_{G_0}[A_0] \leq h(\Pr_{G}[A]) \times h'(\Pr_{G'}[A']) \]

Essence: relate the probability of events in consecutive games

But,

- How do we represent games?
- What adversaries are feasible?
- How do we make a proof hold for any feasible adversary?
What if we represent games as programs?

Games $\implies$ programs
Probability space $\implies$ program denotation
Game transformations $\implies$ program transformations
Generic adversary $\implies$ unspecified procedure
Feasibility $\implies$ Probabilistic Polynomial-Time
**WHILE**: a probabilistic programming language

\[ I ::= V \leftarrow E \quad \text{assignment} \\
| V \leftarrow D \quad \text{random sampling} \\
| \text{if } E \text{ then } C \text{ else } C \quad \text{conditional} \\
| \text{while } E \text{ do } C \quad \text{while loop} \\
| V \leftarrow P(E, \ldots, E) \quad \text{procedure call} \\
\]

\[ C ::= \text{nil} \quad \text{nop} \\
| I; C \quad \text{sequence} \\
\]

Measure monad: \( M(X) \overset{\text{def}}{=} (X \rightarrow [0,1]) \rightarrow [0,1] \)

\[
[[\cdot]] : C \rightarrow M \rightarrow M(M)
\]

\[ [x \leftarrow \{0,1\}; y \leftarrow \{0,1\}] \ m = \]

Probability: \( \Pr_{G,m}[A] \overset{\text{def}}{=} [G] m 1_A \)
PWHILE: a probabilistic programming language

\[ I ::= \begin{array}{ll}
V \leftarrow E & \text{assignment} \\
V \leftarrow \$ D & \text{random sampling} \\
\text{if } E \text{ then } C \text{ else } C & \text{conditional} \\
\text{while } E \text{ do } C & \text{while loop} \\
V \leftarrow \mathcal{P}(E, \ldots, E) & \text{procedure call}
\end{array} \]

\[ C ::= \begin{array}{ll}
nil & \text{nop} \\
I; C & \text{sequence}
\end{array} \]

Measure monad: \( M(X) \overset{\text{def}}{=} (X \rightarrow [0, 1]) \rightarrow [0, 1] \)

\[
\llbracket \cdot \rrbracket : C \rightarrow \mathcal{M} \rightarrow M(\mathcal{M})
\]

\[
\llbracket x \leftarrow \{0, 1\}; \ y \leftarrow \{0, 1\}\rrbracket \ m \ f =
\begin{align*}
\frac{1}{4} f(m[0, 0/x, y]) & \quad + \quad \frac{1}{4} f(m[0, 1/x, y]) \\
\frac{1}{4} f(m[1, 0/x, y]) & \quad + \quad \frac{1}{4} f(m[1, 1/x, y])
\end{align*}
\]

Probability: \( \Pr_{G,m}[A] \overset{\text{def}}{=} \llbracket G \rrbracket m \ 1_A \)
PWHILE: a probabilistic programming language

\[I ::= V \leftarrow E\quad \text{assignment}\]
\[| \quad V \leftarrow \mathcal{D}\quad \text{random sampling}\]
\[| \quad \text{if } E \text{ then } C \text{ else } C\quad \text{conditional}\]
\[| \quad \text{while } E \text{ do } C\quad \text{while loop}\]
\[| \quad V \leftarrow \mathcal{P}(E, \ldots, E)\quad \text{procedure call}\]

\[C ::= \text{nil}\quad \text{nop}\]
\[| \quad I; C\quad \text{sequence}\]

Measure monad: \(\mathcal{M}(X) \defeq (X \to [0, 1]) \to [0, 1]\)

\[\llbracket \cdot \rrbracket : C \to \mathcal{M} \to \mathcal{M}(\mathcal{M})\]

\[\llbracket x \leftarrow \{0, 1\};\ y \leftarrow \{0, 1\}\rrbracket m \mathbb{1}_{x \neq y} =\]
\[
\frac{1}{4} \mathbb{1}_{x \neq y}(m[0, 0/x, y]) + \frac{1}{4} \mathbb{1}_{x \neq y}(m[0, 1/x, y]) + \frac{1}{4} \mathbb{1}_{x \neq y}(m[1, 0/x, y]) + \frac{1}{4} \mathbb{1}_{x \neq y}(m[1, 1/x, y])
\]

Probability: \(\Pr_{G,m}[A] \defeq \llbracket G \rrbracket m \mathbb{1}_A\)
PWHILE: a probabilistic programming language

\[ I ::= \begin{array}{l}
V \leftarrow E \quad \text{assignment} \\
V \leftarrow^\$ D \quad \text{random sampling} \\
\text{if } E \text{ then } C \text{ else } C \quad \text{conditional} \\
\text{while } E \text{ do } C \quad \text{while loop} \\
V \leftarrow P(E, \ldots, E) \quad \text{procedure call}
\end{array} \]

\[ C ::= \begin{array}{l}
\text{nil} \quad \text{nop} \\
I; C \quad \text{sequence}
\end{array} \]

Measure monad: \( M(X) \stackrel{\text{def}}{=} (X \to [0, 1]) \to [0, 1] \)

\[ [\cdot] : C \to M \to M(M) \]

\[ [x \leftarrow^\$ \{0, 1\}; y \leftarrow^\$ \{0, 1\}] m \mathbb{1}_{x \neq y} = \\
0 + \frac{1}{4} + \frac{1}{4}
\]

Probability: \( \Pr_{G,m}[A] \stackrel{\text{def}}{=} [G] m \mathbb{1}_A \)
**PWHILE:** a probabilistic programming language

\[
\begin{align*}
\mathcal{I} &::= \mathcal{V} \leftarrow \mathcal{E} \quad \text{assignment} \\
& | \quad \mathcal{V} \leftarrow \mathcal{D} \quad \text{random sampling} \\
& | \quad \text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} \quad \text{conditional} \\
& | \quad \text{while } \mathcal{E} \text{ do } \mathcal{C} \quad \text{while loop} \\
& | \quad \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) \quad \text{procedure call} \\
\mathcal{C} &::= \text{nil} \quad \text{nop} \\
& | \quad \mathcal{I}; \mathcal{C} \quad \text{sequence}
\end{align*}
\]

Measure monad: \( M(X) \defeq (X \rightarrow [0, 1]) \rightarrow [0, 1] \)

\[
\llbracket . \rrbracket : \mathcal{C} \rightarrow \mathcal{M} \rightarrow M(\mathcal{M})
\]

\[
\llbracket x \leftarrow \{0, 1\}; \ y \leftarrow \{0, 1\} \rrbracket \ m \ 1_{x \neq y} = \frac{1}{2}
\]

Probability: \( \Pr_{G,m}[A] \defeq \llbracket G \rrbracket \ m \ 1_A \)
Untyped vs. typed language

- 1\textsuperscript{st} attempt: untyped language, lots of problems
  - No guarantee that programs are well-typed
  - Had to deal with ill-typed programs

- 2\textsuperscript{nd} attempt: typed language (dependently typed syntax!)
  - Programs are well-typed by construction

\textbf{Inductive } \mathcal{I} : \textbf{Type} :=
\[
\begin{align*}
\text{Assign} & : \forall t, \mathcal{V}_t \to \mathcal{E}_t \to \mathcal{I} \\
\text{Rand} & : \forall t, \mathcal{V}_t \to \mathcal{D}_t \to \mathcal{I} \\
\text{Cond} & : \mathcal{E}_\text{Bool} \to \mathcal{C} \to \mathcal{C} \to \mathcal{I} \\
\text{While} & : \mathcal{E}_\text{Bool} \to \mathcal{C} \to \mathcal{I} \\
\text{Call} & : \forall l, t, \mathcal{P}_{(l,t)} \to \mathcal{V}_t \to \mathcal{E}_l^* \to \mathcal{I}
\end{align*}
\]

\textbf{where } \mathcal{C} := \mathcal{I}^*.

Parametrized semantics: \( \llbracket \cdot \rrbracket : \forall \eta, \mathcal{C} \to \mathcal{M} \to \mathcal{M}(\mathcal{M}) \)
Untyped vs. typed language

- 1\textsuperscript{st} attempt: untyped language, lots of problems
  - No guarantee that programs are well-typed
  - Had to deal with ill-typed programs
- 2\textsuperscript{nd} attempt: typed language (dependently typed syntax!)
  - Programs are well-typed by construction

\textbf{Inductive} $\mathcal{I} : \text{Type} :=$
\begin{itemize}
  \item Assign : $\forall t, \mathcal{V}_t \rightarrow \mathcal{E}_t \rightarrow \mathcal{I}$
  \item Rand : $\forall t, \mathcal{V}_t \rightarrow \mathcal{D}_t \rightarrow \mathcal{I}$
  \item Cond : $\mathcal{E}_{\text{Bool}} \rightarrow \mathcal{C} \rightarrow \mathcal{C} \rightarrow \mathcal{I}$
  \item While : $\mathcal{E}_{\text{Bool}} \rightarrow \mathcal{C} \rightarrow \mathcal{I}$
  \item Call : $\forall l \ t, \mathcal{P}_{(l,t)} \rightarrow \mathcal{V}_t \rightarrow \mathcal{E}_l^* \rightarrow \mathcal{I}$
\end{itemize}
\textbf{where} $\mathcal{C} := \mathcal{I}^*$.

Parametrized semantics: $\llbracket \cdot \rrbracket : \forall \eta, \mathcal{C} \rightarrow \mathcal{M} \rightarrow \mathcal{M}(\mathcal{M})$
Characterizing feasible adversaries

A cost model for reasoning about program complexity

\[ [\cdot]': \forall \eta, \ C \rightarrow (\mathcal{M} \times \mathbb{N}) \rightarrow M(\mathcal{M} \times \mathbb{N}) \]

Non-intrusive:

\[ [G] \ m = \text{bind} ([G]' (m, 0)) (\lambda mn. \text{unit}(\text{fst} mn)) \]

A program \( G \) runs in probabilistic polynomial time if:

- It terminates with probability 1 (i.e. \( \forall m, \ Pr_{G,m}[\text{true}] = 1 \))
- There exists a polynomial \( p(\cdot) \) s.t. if \( (m', n) \) is reachable with positive probability, then \( n \leq p(\eta) \)
Characterizing feasible adversaries

A cost model for reasoning about program complexity

\[ [\cdot]' : \forall \eta, C \rightarrow (M \times \mathbb{N}) \rightarrow M(M \times \mathbb{N}) \]

Non-intrusive:

\[ \llbracket G \rrbracket m = \text{bind} (\llbracket G \rrbracket' (m, 0)) (\lambda mn. \text{unit} (\text{fst} mn)) \]

A program G runs in probabilistic polynomial time if:

- It terminates with probability 1 (i.e. \( \forall m, \Pr_{G,m}[\text{true}] = 1 \))
- There exists a polynomial \( p(\cdot) \) s.t. if \( (m', n) \) is reachable with positive probability, then \( n \leq p(\eta) \)
Program equivalence

Definition (Observational equivalence)

\[
f \equiv_X g \quad \text{def} \quad \forall m_1 \, m_2, \, m_1(X) = m_2(X) \implies f \, m_1 = g \, m_2
\]

\[
\models G_1 \equiv_O G_2 \quad \text{def} \quad \forall m_1 \, m_2 \, f \, g, \, m_1(I) = m_2(I) \land f =^O g \implies [G_1] \, m_1 \, f = [G_2] \, m_2 \, g
\]

Generalizes information flow security.
But is not general enough...

\[
\models \text{if } x = 0 \text{ then } y \leftarrow x \text{ else } y \leftarrow 1 \equiv_{\{x\}} \text{if } x = 0 \text{ then } y \leftarrow 0 \text{ else } y \leftarrow 1
\]
Program equivalence

Definition (Observational equivalence)

\[ f \equiv_{X} g \overset{\text{def}}{=} \forall m_1 \ m_2, \ m_1(X) = m_2(X) \implies f \ m_1 = g \ m_2 \]

\[ \models G_1 \overset{\text{IO}}{\sim} G_2 \overset{\text{def}}{=} \forall m_1 \ m_2 \ f \ g, \ m_1(I) = m_2(I) \land f =_{O} g \implies [G_1] m_1 \ f = [G_2] m_2 \ g \]

Generalizes information flow security.
But is not general enough...

???

\[ \models \text{if } x = 0 \text{ then } y \leftarrow x \ \text{else } y \leftarrow 1 \overset{\{x\}}{\sim} \{x, y\} \text{ if } x = 0 \text{ then } y \leftarrow 0 \ \text{else } y \leftarrow 1 \]
Program equivalence

Definition (Observational equivalence, generalization)

\[ \vdash G_1 \sim G_2 : \psi \Rightarrow \Phi \iff \forall m_1 m_2. m_1 \psi m_2 \Rightarrow \llbracket G_1 \rrbracket m_1 \sim \Phi \llbracket G_2 \rrbracket m_2 \]
Where \( \sim \) is the lifting of relation \( \Phi \) from memories to distributions.

\[
(x = 0) \sim \{x\} (x = 0)
\]

\[
\vdash y \leftarrow x \sim y \leftarrow 0 : =\{x\} \land (x = 0) \langle 1 \rangle \Rightarrow =\{x, y\}
\]

\[
\vdash y \leftarrow 1 \sim y \leftarrow 1 : =\{x\} \land (x \neq 0) \langle 1 \rangle \Rightarrow =\{x, y\}
\]

if \( x = 0 \) then \( y \leftarrow x \) else \( y \leftarrow 1 \sim \)

if \( x = 0 \) then \( y \leftarrow 0 \) else \( y \leftarrow 1 : =\{x\} \Rightarrow =\{x, y\} \)
From program equivalence to probability

Let $A$ be an event that depends only on variables in $O$

To prove $\Pr_{G_1,m_1}[A] = \Pr_{G_2,m_2}[A]$ it suffices to show

- $\vdash G_1 \equiv^I_O G_2$
- $m_1 =^I m_2$
Proving program equivalence

Goal
\[ \Vdash G_1 \simeq^I O G_2 \]

A Relational Hoare Logic

\[ \Vdash c_1 \sim c_2 : \Phi \Rightarrow \Phi' \quad \Vdash c'_1 \sim c'_2 : \Phi' \Rightarrow \Phi'' \]

\[ \Vdash c_1 ; c'_1 \sim c_2 ; c'_2 : \Phi \Rightarrow \Phi'' \]

[R-Seq]

\[ \cdots \]
Proving program equivalence

Goal

\[ \models G_1 \sim^I_O G_2 \]

Mechanized program transformations

- Transformation: \( T(G_1, G_2, I, O) = (G'_1, G'_2, I', O') \)
- Soundness theorem
  \[
  T(G_1, G_2, I, O) = (G'_1, G'_2, I', O') \quad \models G'_1 \sim^{I'}_{O'} G'_2 \\
  \models G_1 \sim^I_O G_2
  \]

- Reflection-based Coq tactic
Proving program equivalence

Goal
\[ \vdash G_1 \simeq^I_O G_2 \]

Mechanized program transformations

- Dead code elimination
- Constant folding and propagation
- Procedure call inlining
- Instruction reordering
- Common suffix/prefix elimination
Proving program equivalence

Goal

\[ \models G_1 \simeq^I_O G_2 \]

A semi-decision procedure for self-equivalence

- Does \( \models G \simeq^I_O G \) hold?
- Analyze dependencies to compute \( I' \) s.t. \( \models G \simeq^{I''}_O G \)
- Check that \( I'' \subseteq I \)
Example

**Game ElGamal\textsubscript{0}**:

\[
x \xleftarrow{\$} \mathbb{Z}_q; \quad y \xleftarrow{\$} \mathbb{Z}_q; \\
(m_0, m_1) \leftarrow A(g^x); \\
b \xleftarrow{\$} \{0, 1\}; \\
\zeta \leftarrow g^{xy} \times m_b; \\
b' \leftarrow A'(g^x, g^y, \zeta); \\
d \leftarrow b = b'
\]

\[
\sim^{\emptyset}_{\{d\}}
\]

**Game DDH\textsubscript{0}**:

\[
x \xleftarrow{\$} \mathbb{Z}_q; \\
y \xleftarrow{\$} \mathbb{Z}_q; \\
d \leftarrow B(g^x, g^y, g^{xy})
\]

**Procedure** \(B(\alpha, \beta, \gamma)\):

\[
(m_0, m_1) \leftarrow A(\alpha); \\
b \xleftarrow{\$} \{0, 1\}; \\
b' \leftarrow A'(\alpha, \beta, \gamma \times m_b); \\
return b = b'
\]
The Fundamental Lemma of Game-Playing

Fundamental lemma
If two games $G_1$ and $G_2$ behave identically in an initial memory $m$ unless a failure event $A$ fires, then

$$|\Pr_{G_1,m}[A] - \Pr_{G_2,m}[A]| \leq \Pr_{G_1,2}[F]$$
The Fundamental Lemma of Game-Playing

\[
\begin{align*}
\text{Game } G_1 & : \\
& \ldots \\
& \text{bad } \leftarrow \text{true}; \ c_1 \\
& \ldots \\
\text{Game } G_2 & : \\
& \ldots \\
& \text{bad } \leftarrow \text{true}; \ c_2 \\
& \ldots
\end{align*}
\]

- \( \Pr_{G_1,m}[A \land \neg \text{bad}] = \Pr_{G_2,m}[A \land \neg \text{bad}] \)
- \( \Pr_{G_1,m}[\text{bad}] = \Pr_{G_2,m}[\text{bad}] \)

**Corollary**

\[
|\Pr_{G_1,m}[A] - \Pr_{G_2,m}[A]| \leq \Pr_{G_1,2}[\text{bad}]
\]
Wrapping up

**Contributions**
- Formal semantics of a probabilistic programming language
- Characterization of probabilistic polynomial-time programs
- A Probabilistic Relational Hoare logic
- Mechanization of common program transformations
- Formalized emblematic proofs: ElGamal, FDH, OAEP

**Perspectives**
- Overwhelming number of applications: IB, ZK proofs, ...
- Computational soundness of symbolic methods and information flow type systems
- Verification of randomized algorithms
Some statistics

- 6 persons involved
- CertiCrypt: 30,000 lines of Coq, 48 man-months
- Full Domain Hash: 2,500 lines of Coq, 4 man-months (for a person without experience in CertiCrypt)
Questions
ElGamal encryption

\[
\Pr_{\text{ElGamal}}[b = b'] - \frac{1}{2} = |\Pr_{\text{DDH}_0}[d] - \Pr_{\text{DDH}_1}[d]|
\]
Observational equivalence

\[ \models G_1 \sim G_2 : \psi \Rightarrow \Phi \overset{\text{def}}{=} m_1 \psi m_2 \Rightarrow [G_1] m_1 \sim_{\Phi} [G_2] m_2 \]

**Lifting**

\[
\text{range } P \mu \overset{\text{def}}{=} \forall f, (\forall a, P a \Rightarrow f a = 0) \Rightarrow \mu f = 0
\]

\[
\mu_1 \sim_{\Phi} \mu_2 \overset{\text{def}}{=} \exists \mu, \pi_1(\mu) = \mu_1 \land \pi_2(\mu) = \mu_2 \land \text{range } \Phi \mu
\]
Small-step semantics

\[
\begin{align*}
(nil, m, [ ]) & \rightsquigarrow \text{unit } (nil, m, [ ]) \\
(nil, m, (x, e, c, l) :: F) & \rightsquigarrow \text{unit } (c, (l, m.\text{glob})\{[e] m/x\}, F) \\
(x \leftarrow p(\bar{e}); c, m, F) & \rightsquigarrow \text{unit } (E(p).\text{body}, (\emptyset\{[\bar{e}] m/E(p).\text{params}\}, c, m, F) \\
(\text{if } e \text{ then } c_1 \text{ else } c_2; c, m, F) & \rightsquigarrow \text{unit } (c_1; c, m, F) \\
& \quad \text{if } [e] m = \text{true} \\
(\text{if } e \text{ then } c_1 \text{ else } c_2; c, m, F) & \rightsquigarrow \text{unit } (c_2; c, m, F) \\
& \quad \text{if } [e] m = \text{false} \\
(\text{while } e \text{ do } c; c', m, F) & \rightsquigarrow \text{unit } (c; \text{while } e \text{ do } c; c', m, F) \\
& \quad \text{if } [e] m = \text{true} \\
(\text{while } e \text{ do } c; c', m, F) & \rightsquigarrow \text{unit } (c', m, F) \\
& \quad \text{if } [e] m = \text{false} \\
(x \leftarrow e; c, m, F) & \rightsquigarrow \text{unit } (c, m\{[e] m/x\}, F) \\
(x \leftarrow d; c, m, F) & \rightsquigarrow \text{bind } ([d] m)(\lambda v. \text{unit } (c, m\{v/x\}, F))
\end{align*}
\]
Denotation

\[ [S]_0 \overset{\text{def}}{=} \text{unit } S \quad [S]_{n+1} \overset{\text{def}}{=} \text{bind } [S]_n [\cdot]_1 \]

\[ [c] \; m : M(M) \overset{\text{def}}{=} \lambda f. \; \sup \{ [c, m, [\cdot]]_n f |_{\text{final}} \mid n \in \mathbb{N} \} \]